ECE 250 Project 1 Playlist
Design Document

Anthony Xu, UW User ID:a68xu

Oct 12 2022

Overview of classes:

Classes used:

- Node Class
- LinkedList Class
- Dequeue Class(inherited from LinkedList Class)

Node Class

Description:

The Node class represents a single Node which holds four fields of data, one for the URL-name and one for the URL. The other two fields are pointers to the previous and next Nodes.

LinkedList Class

Description:

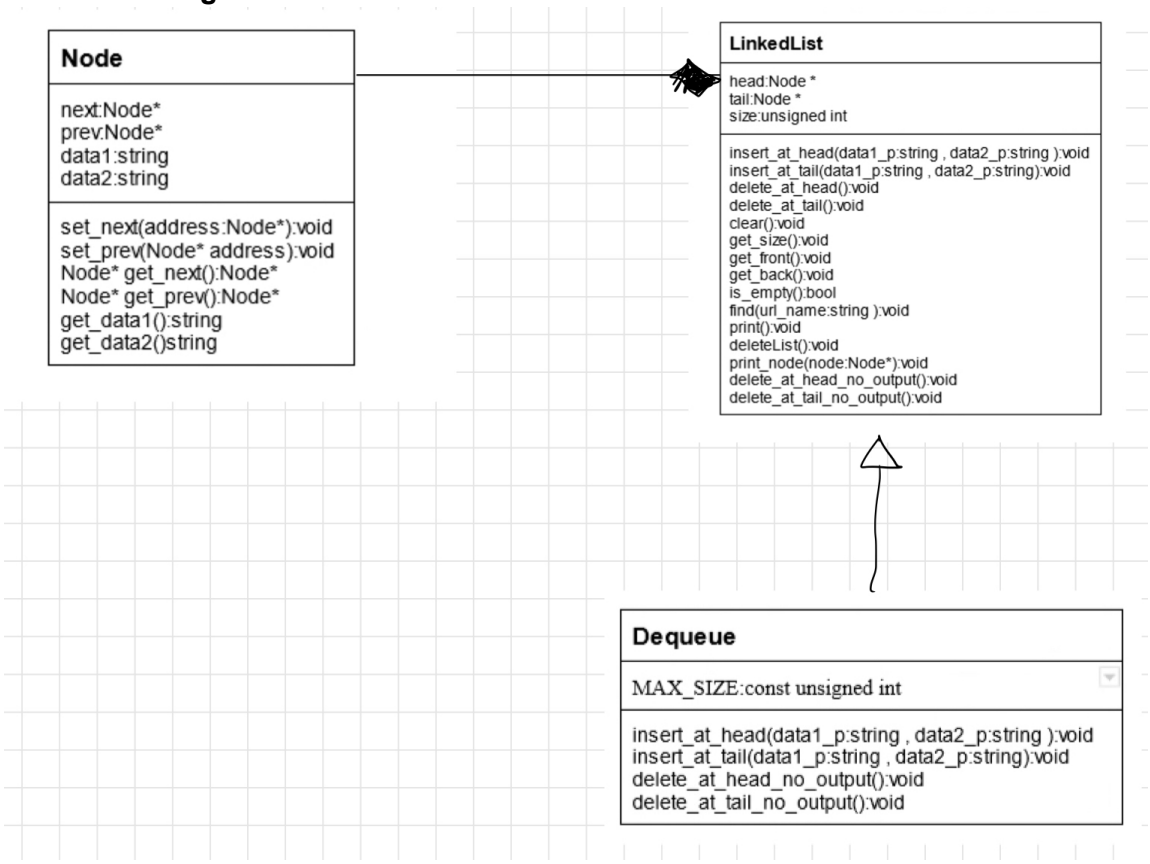
The LinkedList class has most of the functionalities of a doubly linked list. The most important member variable and LinkedList object is the address of the head and tail node. As well as a list size to keep track of.

Dequeue Class

Description:

The Dequeue class inherits from the LinkedList class and adds functionalities. An extra MAX_SIZE variable is added, and initialized with the overloaded constructor. The push_front and pop_front functionalities are also reimplemented to allow for deletion at the other end when inserting.

UML Class diagram



From: http://www.cs.utsa.edu/~cs3443/uml/uml.html

Details on design decisions

Node Class

```
  private:
    Node* next;
    Node* prev;
    string data1;
    string data2;

  public:
    Node(string data1, string data2);
    Node();
    ~Node();
    void set_next(Node* address);
    void set_prev(Node* address);
    Node* get_next();
    Node* get_prev();
    string get_data1();
    string get_data2();
```

- The default constructor is empty
- The main constructor initializes data1 and data2 variables. Although this project does not allow for modification on individual URLs, I did not use a const variable(there is no function that can allow for modification)
- The default deconstructor is empty, the node should be removed in memory when the delete keyword is used
- To allow for modification of the LinkedList(insertion,deletion) set_next and set_prev data are used. A const keyword could be added for the parameter that is passed in since no modification should be made to the parameter.
- To allow for safer access of member variables when traversing the LinkedList and getting Node data, get_next(), get_prev(), get_data1(),get_data2()functions are used.

LinkedList Class

```
protected:
    Node *head;
    Node *tail;
    unsigned int size;

public:
    LinkedList();
    ~LinkedList();
    void insert_at_head(string data1_p, string data2_p);
    void insert_at_tail(string data1_p, string data2_p);
    void delete_at_head();
    void delete_at_tail();
    void clear();
    void get_size();
    void get_front();
    void get_back();
    bool is_empty();
    void find(string url_name);
    void print();
    void deleteList();
    void print_node(Node* node);
    void delete_at_head_no_output();
    void delete_at_tail_no_output();
```

- Default constructor initializes the head and tail nodes and hook them up by modifying the next and prev values of the two nodes
- The deconstructor is defined, it first clears the list, deleting all nodes, then exits(the deconstructor would free the space occupied by the head and tail node, as well as size variable)
- For each of the functions with parameters, the variable should be passed in as const variable, since they are not modified
- I added an additional function print_node(Node* node), which allows for more convenient printing of information when deleting, finding a node,calling front and back

Dequeue Class

```
private:
        const unsigned int MAX_SIZE;
    public:
        Dequeue(unsigned int dequeue_Max_Size);
        void insert_at_head(string data1_p, string data2_p);
        void insert_at_tail(string data1_p, string data2_p);
        void delete_at_head_no_output();
        void delete_at_tail_no_output();
```

- The constructor of the inherited class is overloaded, which allows for initializing the MAX_SIZE variable (the parameter should be passed in as const)
- The destructor is not defined since when we delete a dequeue object, the deconstructor of the base class will be automatically called
- The insert_at_head(string data1_p,string data2_p) and insert_at_tail(string data1_p,string data2_p) functions are overloaded from the inherited class, which allows them to still insert when the list is full by popping a node at the other side of the list. The parameters should be passed in as const since we are not modifying it

Test Cases

1. Test all operations by it self with proper corresponding test cases
2. Insert until the list is full and then try to insert again, when doing this, check if the list size is correct
3. Try to delete when there is no nodes in the list
4. after inserting till the list is full, insert again, and print the list to see if the nodes at the other side of the list is deleted, and new nodes are inserted. Delete nodes at the front and back, print to see again
5. Try all operations when the list is full
6. Try all operations after the list is full and have gone through additional insertions and deletions
7. Try all operations when the list is empty
8. Try all operations when the list has one node
9. Try all operations when the list has multiple nodes

Run Time

- Push_front
- Push_back
- Pop_front
- Pop_back
- Front
- Back
  For the above functions, we can achieve the operation in constant timeO(1), this is because there is no need to traverse the whole list to perform those operations. Since we have address of the head and tail node which allows us to perform actions at the ends of the dequeue
- Empty
  For the empty function, we can know if the list is empty by checking whether the list size is 0, which takes constant time. The list size is kept track of by incrementing and decrementing it whenever we insert or delete a node.
- clear
  The clear command takes O(n) time since I traversed the whole list from the tail to head order to delete the nodes one by one. This operation is proportional to the input size(list size) “n”
- print
  The print command takes O(n) time since we also need to traverse the whole list from tail to head in order to print all the nodes